ENABLING SEAMLESS WIDE AREA MIGRATION OF ONLINE GAMES

BY

VIRAJITH JALAPARTI

THESIS

Submitted in partial fulfillment of the requirements
for the degree of Master of Science in Computer Science
in the Graduate College of the
University of Illinois at Urbana-Champaign, 2013

Urbana, Illinois

Adviser:

Assistant Professor Matthew Caesar

Abstract

Highly interactive network applications such as online games are rapidly growing in popularity and make up a multi-billion dollar industry. However, they remain challenging for game providers to support due to their inherent need for low latency, unpredictability in workloads, the need to support application-specific requirements (e.g., players may wish to play with certain other players already in the game) and their scale.

Cloud computing has proven to be a useful alternative to host a variety of online services. While eliminating the costs of owning the physical infrastructure and maintenance, cloud services provide several benefits such as providing agility and reliability for applications. However, existing cloud computing facilities are insufficient for hosting games due to their lack of application-specific primitives and the stringent requirements of online games.

In this work, we first perform an experimental study using real world deployments of several popular games to understand the characteristics of online games. We find that games can benefit from *dynamic allocation* and *seamless migration* of resources across the wide-area to operate efficiently and provide the best user experience. To meet these requirements, we propose *SMOG*, a framework that dynamically migrates game servers to their optimal location, using orchestrated route control to optimize the network path to the server. Through deployment of a prototype implementation on a Tier-1 ISP's backbone and a user study, we found SMOG can decrease average end-user latency by up to 60% while performing migration in a manner transparent to game players. We further find that SMOG provides benefits under a variety of realistic operating conditions, tolerating variation in application load, network latency and bandwidth available for migration. While we focus on online games in this work, SMOG is general enough to be used for a variety of latency-sensitive interactive applications such as video conferencing and interactive video streaming.

To my parents.

Acknowledgments

I would like to thank my adviser, Matthew Caesar, for his advice and mentorship which has been greatly instrumental for my growth as a researcher over the past few years. Matt has been an amazing advisor – I learned a great deal from him about finding innovative problems and questioning oneself during the process. He was very instrumental in helping me learn how I could present my research.

I would like to thank Kobus van der Merwe (now at University of Utah), Jeffery Pang and Seungjoon Lee at AT&T Labs - Research, who were very instrumental in the project that led to this thesis. Part of this work was done while I was interning with them at AT&T during the Summer of 2010. Their support was very helpful for deploying our system over production networks. Their continued help and feedback was very important for the success of this project. I am also grateful to all my fellow interns at AT&T who made my summer there, very enjoyable.

I would like to thank all the students in the Systems and Networking group at UIUC who had helped me with the user study. I would also like to thank them for creating a great atmosphere in our lab – the discussions with my fellow students played an important role in expanding my understanding of various areas of networking.

I would like to thank all the TPC members of ACM NetGames 2012 where an earlier version of this thesis was published – this workshop provided me a forum for publicizing this work and understanding the perspective of game developers and researchers. It gave me an amazing opportunity of traveling to Europe, which was really enjoyable.

Finally, I would like to thank my family and friends who has always been there and supported me during my graduate school.

Table of Contents

Chapter 1

Introduction

Cloud computing has emerged with enormous success as a new paradigm for distributed computing. In this model, data centers provide dynamically allocated resources for applications, owned by different users. This computing model offers significant economic advantages of scale for purchasing, operations, and energy usage since the number of idle machines can be reduced by sharing hardware across multiple users. Cloud services also offer a potential advantage for security and management as such administrative tasks can be performed promptly by the cloud provider, and the cost of system administration can be amortized over many machines.

However, there is a particular class of applications that are difficult to support in cloud environments: highly interactive network applications, such as online games. The online gaming market is a rapidly growing multi-billion dollar industry [1] with hundreds of millions of players [2]. These applications face several unique challenges. Online games have stringent quality of service (QoS) demands, where even small increases in latency can interfere with game play [3–6]. Hence, they have strong requirements on where servers can be placed. At the same time, the unpredictability of network QoS coupled with the difficulty in predicting where a game will become popular make it hard to statically place these services. Even worse, unlike web users, players of certain types of games like Counter Strike: Source [7], Quake III [8], etc. often have strong affinity to particular servers (see Chapter 2). This makes it difficult to use traditional load balancing and replication mechanisms (e.g., CDNs) to serve game clients. While application specific schemes have been proposed for network games to deal with some of these problems (e.g., by partitioning state [9, 10]), they require substantial changes in the way game developers architect games.

Such challenges are particularly unfortunate for two broad classes of games that can obtain substantial benefit from shared cloud infrastructure to meet their needs: (a) games that are hosted by individual players such as Counter Strike: Source, and (b) the majority of games that have not yet achieved a player population large enough to justify deploying a dedicated server infrastructure around the world. Shared cloud services would reduce their operating costs and improve players' gameplay experiences. Similar challenges pre-

vent other highly interactive applications such as video conferencing and interactive video streaming [11], from reaping the full benefits of the cloud.

In this work, we explore the primitives which would enable cloud services to host online games while meeting their strict constraints. To this end, we perform an extensive study of several online games to understand their basic requirements. In particular, we find that providing *dynamic migration* of cloud computing resources can simplify the problem of hosting game servers. We propose a platform for *Seamless Migration of Online Games* or SMOG, which dynamically optimizes geographic server (or virtual machine) placement while minimizing the observable effects of live server migration. To effectively address this problem, we take the relatively unexplored approach of *integrating cloud services with wide area network services*. While making cloud services network-aware has been explored in previous work [12–14], they either require modifications to the network components or applications being supported, or experience several seconds of downtime. SMOG overcomes these limitations of earlier work and provides a platform for hosting latency-sensitive applications. It requires no application-level modifications and uses the capabilities of the hypervisors in virtualized systems to achieve its goals. SMOG is especially useful for games where players may select the server that they play on for reasons other than latency. These reasons include skill-level, social, or historical preferences. This server selection model, which does not rely on a matchmaking entity, is used by a large number of games like Counter Strike, Quake etc. While our techniques can be used for any online game, they are particularly useful for games in which players choose the server and is complementary to other approaches to match a player to a nearby server.

SMOG can be deployed across multiple data centers which can potentially have different network service providers. While we focus on server migration, game service providers may also wish to use replication to scale up and down their services to adapt to long-term load changes. These techniques are orthogonal to our work, and SMOG can make use of traditional mechanisms to support these goals. While this work's focus is online games, SMOG is general enough to be used for a variety of latency-sensitive interactive applications.

1.1 Contributions

We make three primary contributions in this work:

- We conduct a measurement study showing that there is substantial opportunity for both resource savings and improvement in end-user latency, if game servers could

be dynamically relocated in the wide area. Compared to a static server placement, average user latency can be decreased by up to 60% using dynamic server relocation.

- We designed, implemented and deployed SMOG on data centers connected to a Tier-1 ISP, enabling seamless game server migration in the wide area. In contrast to previous game platforms, SMOG does not require any modifications to game servers or clients, does not require the use of any application-level middleware and thus, allows game developers to continue using traditional game architectures.

- We show, using an user study, that the short interruption caused by SMOG's migration in the wide area is not noticeable to game players. Since previous work has shown that reducing latency improves both subjective and objective player satisfaction measures [3–6], using SMOG's migration primitive to optimize player latency strictly improves the gameplay experience. We also show that seamless migration is feasible with SMOG in a range of realistic operating conditions. While most of the evaluation results presented in this work are for Quake III [8], we observed similar results for two other games – Counter Strike: Source [7] and Plane Shift [15].

Chapter 2

Study of Online Games

Several measurement studies have been performed to understand the latency requirements for online games [3–6, 16–19]. These studies focus on characterizing the effects of increased latency on players' performance. However, they assume that the server position is fixed, which limits the flexibility in hosting game servers. In this chapter, we will try to understand the latency gains that can be achieved with the ability to move game servers. As background, we first describe several popular online games and their requirements (§2.1). Then, we present the results of a measurement study of popular online games (§2.2), which motivates our design of SMOG.

2.1 Background: Online Games

Two of the most popular genres of online games today are First Person Shooter (FPS) and Massively Multi-player Online Games (MMOGs). FPS games such as Quake III [8] and Counter Strike: Source (CSS) [7] are typically fast-paced and thus, have very stringent latency requirements. The game client sends updates to the server every few tens of milliseconds over UDP. They typically have little persistent state and are limited to a few tens of players per game server. On the other hand, MMOGs run at a much slower pace (updates sent every few 100s of millseconds), involve several thousands of players on the same game server and store persistent information for every player in the game world. These include games such as Second Life [20] and World of Warcraft [21].

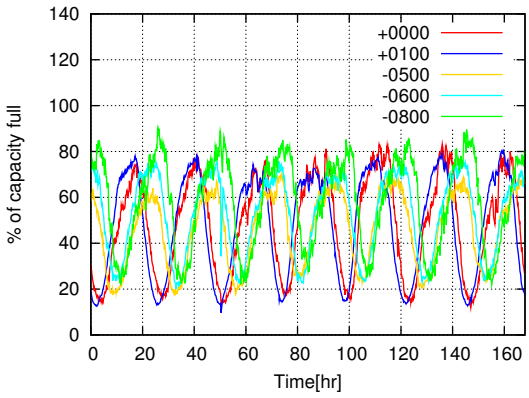
Games such as World War II Online [22] and PlanetSide [23] lie in between these two genres – they have strict latency requirements similar to FPS games, and store persistent state and support hundreds or thousands of players like MMOGs. Further, there has also been much recent interest in delivering games *on demand* so that players do not have to download any client software at all (e.g., games on OnLive [11]). This entails a game server in the cloud that receives keystrokes and sends only a generic video stream of the game to each client. As real-time video delivery is demanding, these games are also very latency sensitive.

It has been widely known that having low latency between the players and the game server is essential for a good user experience (ideally, under 50 ms [3, 6]). Today, game providers typically achieve such low latency by replicating servers throughout the world, either by deploying dedicated servers in global data centers or by encouraging players to host servers themselves. However, dedicated server replication is expensive and requires over-provisioning to handle the peak load in each geographic region. Player-driven deployments have unpredictable QoS and increases the likelihood of cheating because gaming services do not typically have control over client-hosted software [24]. More importantly, in both scenarios, players often play on non-local servers for a number of reasons – their friends are present on a remote server, remote servers have more players closer to their skill level, local servers have few players etc. Players can join and leave these games at any time and each server runs *forever*. Thus, the group of players playing on a server typically forms and evolves organically.

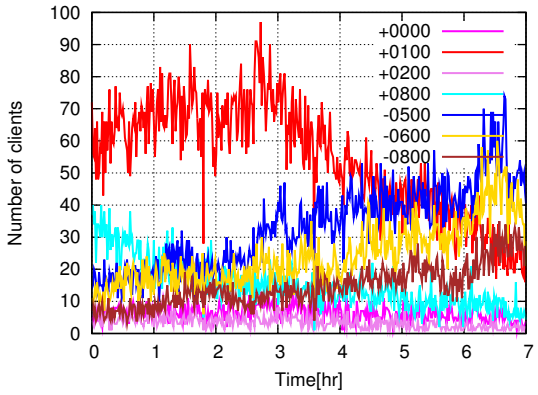
In this work, we focus on games where currently players may want or have no choice but to play on *static* servers that are distant in terms of network latency. This is especially important for (a) games that are hosted by individual players, like CSS, and (b) the majority of games that have not yet achieved a player population large enough to justify deploying a dedicated server infrastructure around the world. We develop a platform called SMOG, which targets such games so they can take advantage of cloud services to reduce their operating costs and improve players' gameplay experiences. While SMOG can support many different types of games, we focus on First Person Shooter (FPS) games because they have the most stringent QoS requirements. In the rest of this thesis, we use the term *game server* to refer to the virtual machine running the application processes for the game.

2.2 Measurement Study

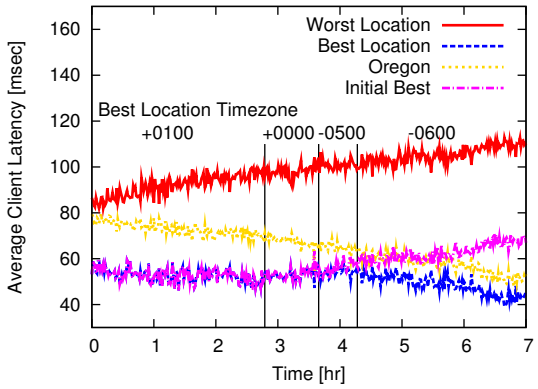
In this work, we argue that *dynamic, seamless server migration* can significantly simplify hosting of online games in cloud computing environments while improving the experience of game players. This is motivated by the insights garnered from a measurement study we performed using Counter Strike: Source (CSS), one of the most popular First Person Shooter (FPS) games with 1.716 billion player minutes per month [25]. (We observed similar results for Quake III [8] and Unreal Tournament [26]). Our study, whose results we discuss below, finds several unique features of games that motivate the key design decisions behind SMOG's architecture.



(a) Variation of occupied capacity with time



(b) Number of players connected to mshmro.com divided across timezones



(c) Variation of average player latency for different server locations

Figure 2.1: Timezone level statistics (UTC=+0000)

2.2.1 Data Sets

To understand the characteristics of online games, we collected two different types of data sets. First, to understand how the load on game servers changed over time and information about players' sessions, we obtained the list of 1000 most popular CSS servers from Game Tracker [27]. Using QStat [28] to obtain server information, we probed these servers at 10-minute intervals over a period of one week, starting from 9:30PM on June 8, 2010 EDT. QStat provides real-time details about the game server including detailed information such as number of players present, player statistics etc. While this provides us information regarding player sessions on the game server, it does not allow us to determine the geographical origin of the players. To obtain this information, we used a 7-hour `tcpdump` trace collected on April 11, 2004 starting at 9AM PDT from mshmro.com [29], a long running popular CSS server located in Portland, Oregon, USA. This contains a detailed packet trace of all connections to this server over the specified time period. It allows us to infer the IP address of the players on the server and in turn their locations using IP geolocation

services [30].

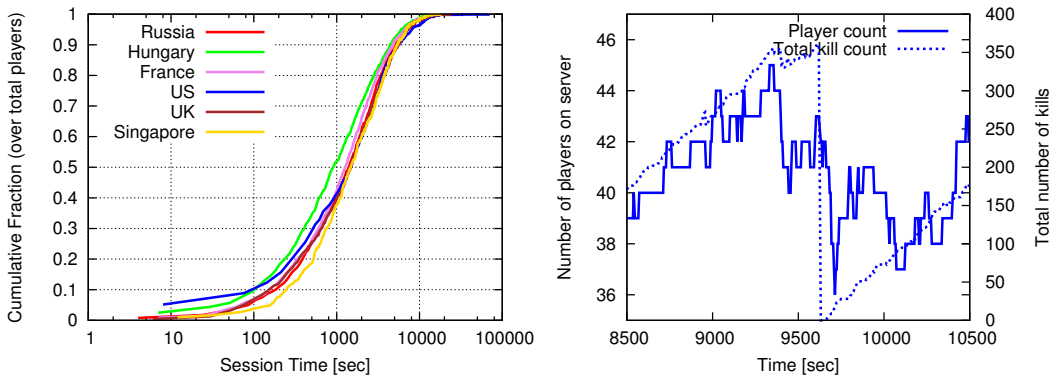
2.2.2 Results

Using the traces collected above, we observe the following three main characteristics of online games, which led us to conclude that games can significantly benefit from dynamic, seamless server migration.

Games need *dynamic*, not *static* allocation of resources: Using the server statistics obtained using Qstat from the top 1000 most popular CSS servers, we make two observations. First, *server loads change greatly over time*. Figure 2.1a shows the *load* on servers in various time zones over the 7-day period. We define load as the ratio of the total number of players in a time zone to the sum of the maximum capacities of the servers in that time zone. We observe a strong diurnal pattern with peak loads of 4-5 times the troughs. Second, we also find that *peak loads shift over time according to the timezones of the servers' geographic locations.* For example, load peaks in the eastern hemisphere occur when load in the western hemisphere is lowest, and vice versa.

This suggests that *game providers can benefit from dynamic hosting of resources*. Static server placement would result in servers located in non-peak areas having wasted capacity. With dynamic placement, game providers could migrate servers from lightly loaded regions into peak regions to cope with load, and to react to dynamics in network path quality and server utilization. Mechanisms like turning off servers under low load do not work because even under low load conditions game servers can have some users playing on them. We note that the peak game playing time is in the evening hours of each time zone, whereas data center usage for businesses typically peaks during day time hours [31]. This means that using the same data centers to host both business and game applications would improve their overall utilization.

Games need *migration*, not *replication* of resources: Next, studying the 7-hour long trace from mshmro.com, we find that *users may express a strong affinity to specific servers*. Although it is well known that players prefer servers that have lower latency [3], our measurements show that geographic and network distances are not the sole criteria in server selection. For example, Figure 2.1b shows that players from all over the world (i.e., several time zones) choose to play on the Oregon CSS server even though there exist servers, closer to them. Users may prefer to play on servers that their friends play on, host specific scenarios, are popular, or contain players with similar skill levels as themselves. Further, in games with persistent state, players may always choose the same server to retain their personal history.



(a) CDF of player session times

(b) Change in kill count of players on a server

Figure 2.2: Server level statistics

Further, we find that *the optimal server position varies dynamically* and *server migration can substantially improve client performance*. Figure 2.1c shows the estimated average client latency given different server placements. For this experiment, we use data from the trace collected from mshmro.com to determine the set of clients connected to the game server. We assume the game server can be located at multiple physical locations corresponding to multiple data centers in a Tier-1 ISP. We estimate the latency experienced by clients using their *air-mile distance* to the server [32]. The air-mile distance is obtained by mapping the IP addresses of the clients and servers to a *(latitude,longitude)* coordinate pair using IP geolocation [30] and then finding the *great-circle distance* between the two coordinate pairs.

In this experiment, we use the Best Location for the server at time t as that physical location which has the minimum average latency to all the users connected to the server at t. Vertical lines in Figure 2.1c indicate a change in the time zone of the best server location. By using migration (Best Location in Figure 2.1c), the average client latency to the server can always be kept close to 50ms, the highest latency that doesn't degrade FPS player performance [33] under any type of game play. In contrast, the two static server placements (Oregon and Initial Best) have up to 60% higher latencies, compared to the Best Location. We expect the worsening trend would continue after hour 7 since the trace does not cover the hours when load from the western U.S. and Asia would peak. The Worst Location line shows that a static placement could be suboptimal by more than 100%.

These results suggest that *game providers can benefit from migration of resources*. Unlike traditional services that use replication and can redirect new user requests to the appropriate (for example, closest) replica, a user's affinity to a particular online game server means that existing servers may need to be moved closer to clients to achieve the best user experience. Also, since the majority of load arises from a relatively small geographic re-

gion at any point in time, it may be possible to place a single server to satisfy most of the users that have an affinity for it. Thus, replication of servers in multiple geographic regions would waste resources and would not satisfy user affinities to specific servers, unless they were nearby.

Games need *seamless*, not *paused* migration: To understand how long a user is connected to a particular server, we studied the session times of players, using the above week long trace from the top 1000 servers. Here, we make two observations. First, *some session durations are very long*. Figure 2.2a shows the cumulative fraction of the session times of the players connected to the most popular servers in different countries over a period of 24 hours. While the median session time is around 20 minutes, some players go on to play for several hours. Second, we find that *session durations have large variance*, ranging over several orders of magnitude. These two observations suggest that *migration must be performed while users are playing*, introducing the need for *seamless migration*. The existence of multi-hour session durations means that traditional *dryout* techniques used for non-gaming services (redirecting new users to another server and waiting for the server to empty) may take too long to react to diurnal patterns or flash crowds in online games. Moreover, since session durations have large variance, it is hard to find specific points in time where migration can be performed with minimal interruption (since most users are typically in the middle of their game at any point in time). Thus, a general-purpose online game hosting service should support seamless migrations.

That said, some games possess properties that can assist in making this seamless migration easier to achieve. For example, we find that *some games undergo periods of decreased user activity*. For example, CSS takes place in multiple levels. After every level, players return to their initial positions, and the game map and statistics are reset. During this time, players are static and hence, transient packet loss due to migration is less likely to be directly perceivable to players. We note that these times can be detected without modifying the game. Figure 2.2b shows how the kill count of players and the number of players in a CSS server in San Francisco change over a period of 2000 seconds. This graph is deduced from a trace that was collected external to the game using QStat [28]. The sudden decrease in kill count half-way through indicates a level change. Designing game hosting services to be application aware can help identify such periods during which game servers can be transparently relocated. We note that such application properties are not necessary for providing seamless migration using SMOG (as shown by the user study in §5.1) but can be useful to further improve its performance.

Chapter 3

Supporting Online Games on Cloud Platforms

To meet the unique requirements of hosting online game servers in a cloud environment, as determined by our measurement study, we designed a platform for *seamless migration of online games* called SMOG. SMOG needs to address two key problems: First, to provide the best service to the application, SMOG needs to seamlessly migrate services across the *wide area*. Migrating interactive application services across the wide area presents new challenges, which are not addressed by traditional virtual machine migration techniques. For example, migrating servers across IP domains introduces the danger of disconnections and routing convergence issues, and migrating servers across long distances can increase latency with potential downtime due to migration. To address this, SMOG provides *network support* for virtual machine migration, coordinating application migration with route control in a manner that enables fast migration across the wide area with minimal service disruption.

Second, SMOG needs to determine when and where to migrate the application services. This has to be done while ensuring game servers are placed in locations that provide the best service to users. At the same time, such migration should not be performed too frequently to avoid oscillations and unnecessary overhead. To address this, SMOG solves an assignment problem to dynamically relocate interactive applications in the wide area. These algorithms execute on the SMOG Control Framework, which collects state about application-specific performance metrics, determines when a particular application service needs to be migrated, and instructs the SMOG infrastructure to perform that migration.

In the rest of this chapter, we provide an overview of SMOG's architecture and then discuss the details of how SMOG addresses the above challenges.

3.1 SMOG Architecture

The architecture of SMOG is depicted in Figure 3.1. The cloud service provider deploys a set of *hosting platforms* connected to a wide area network (WAN). For example, an ISP network operator may integrate cloud computing functionality into its backbone by deploying

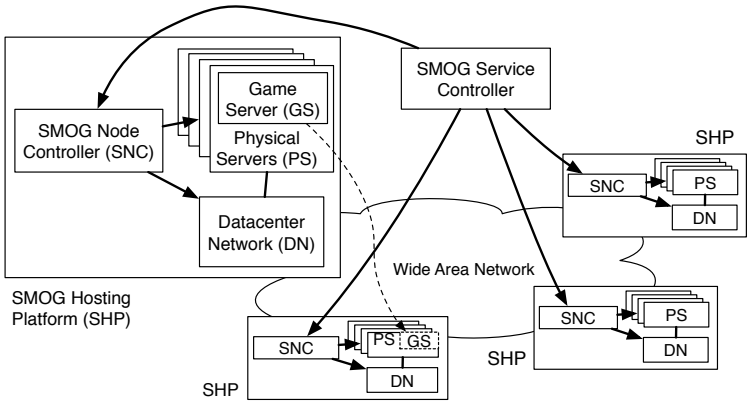


Figure 3.1: SMOG provides a distributed platform for online games, consisting of hosting facilities deployed across the wide area.

hosting platforms in its points-of-presence. These hosting platforms act as facilities where application processes (e.g., game servers) may be hosted. In our design, each application process runs within its own virtual machine, and migration is performed by the hypervisor. As shown in Figure 3.1, each hosting platform consists of (i) a number of *physical servers* running virtualization software that supports virtual machine migration, (ii) *datacenter networking* equipments that provide connectivity within the platform as well as connectivity to the WAN, and (iii) a *SMOG node controller* which is responsible for local orchestration of compute and networking resources to realize SMOG functionality. SMOG node controllers in turn are collectively orchestrated by a *SMOG Service Controller* which makes migration decisions in order to realize the service objectives of the hosted application. Collectively the SMOG node and service controllers constitute the SMOG Control Framework.

To enable SMOG functionality, the SMOG node controllers expose an API to monitor and manipulate virtual machine and networking state. For example, using this API, VM migration is controlled. It is also used to manipulate routes, re-direct traffic, and observe traffic patterns and routing tables within the network in order to support migration of servers. The API also allows the service controller to collect application specific performance metrics (for example, network latency to clients in the case of online games) from the hosting platforms. Such information is collected by the virtualization software, at each of the hosting points, by monitoring client connections to the servers. This performance information is used to make decisions regarding when servers should be migrated, and which new hosting platforms they should be migrated to.

To begin a migration, the service controller uses the node controller API to instruct the virtualization software regarding when and where the process should be migrated to. To coordinate this change with the network, the service controller also sends instructions to

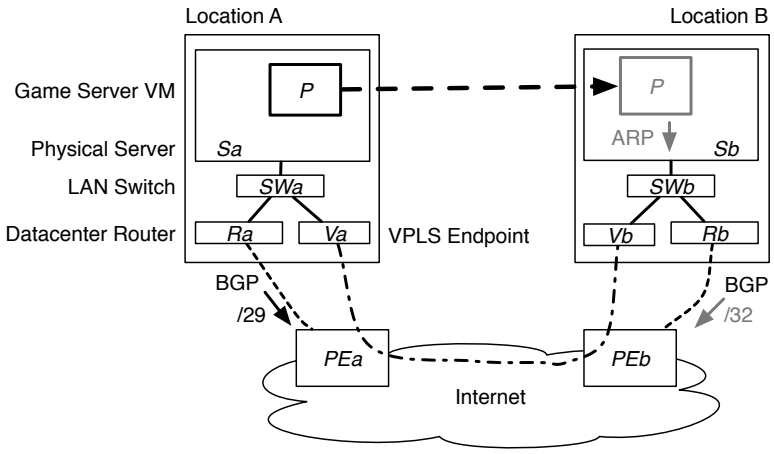


Figure 3.2: SMOG Migration

the API to indicate any changes to routing state in order to ensure that the packets destined to the migrated server reach it.

3.2 Enabling Wide Area Migration

To adapt to changing application workloads and provide better user experience, SMOG needs to perform server migration between different hosting platforms i.e., across LANs, across IP subnets, and potentially across Autonomous Systems (ASes). Unfortunately, traditional virtual machine migration techniques (e.g., Xen [34]) target migration within a single Ethernet-based IP subnet, as opposed to WANs that span large geographic regions, multiple routing protocols, and multiple IP subnets. Further, these techniques were designed for LANs where a virtual machine (VM) can retain its IP address after its migration. Such mechanisms cannot directly be used in wide area networks (i.e., across hosting points) as the IP address of the VM might have to be modified to comply with network addressing. Such changes in the IP addresses of the servers are not acceptable for interactive applications since it would require all current sessions to be terminated, interfering with the user's experience.

To address this, SMOG leverages *integrated network support* to coordinate migration of an application server, in a manner that minimizes the downtime (period of disconnectivity to the server) experienced by the application users. In particular, SMOG coordinates server migration with *network route control*. In this section, we describe how SMOG works when the hosting points are within a single AS. The inter-AS migration is discussed in §4.3.

Figure 3.2 depicts the mechanisms involved in migrating an application server (P) from

location A to location B. Each application server running on SMOG is allocated a small IP subnet (e.g., a /29). Under normal operating conditions (i.e., when a migration is not under way), this prefix would be advertised by the datacenter router (i.e., *Ra* in our example) via BGP to a provider edge router (*PEa*) to allow the application server to be reached from the Internet. The application server would similarly use *Ra* as its default gateway in order to reach clients in the Internet.

SMOG utilizes a virtual private LAN service (VPLS) (*Va* to *Vb* via *PEa* and *PEb*), to facilitate live virtual machine migration between physical servers Sa and Sb respectively [35].[1] VPLS is a MPLS-based virtual private network (VPN) and essentially interconnects the LAN switches SWa and SWb at layer two. This creates a broadcast domain between the two sites, allowing IP addresses to move at will between the two sites and still be reachable from anywhere in the broadcast domain. This in turn enables the use of "normal" LAN VM migration techniques [34] to migrate the application server (*P*) from Sa to Sb. Further, after migration, *P* issues a gratuitous ARP response and will be still reachable from the Internet via location A i.e., packets destined for *P* will reach *PEa*, which is still advertising the /29 prefix, and then follow the path to *P* via *Ra*, SWa, *Va*, *PEa* (now encapsulated in a VPLS frame), *PEb*, *Vb*, SWb, Sb and *P*. After migration *P* will still be using *Ra* as its default gateway, so that packets from *P* will follow the same path in reverse.

Having traffic follow this "dogleg" path is of course suboptimal, and the SMOG service control corrects this by performing the following two route control actions. First, the default gateway for *P* is updated to *Rb* so that traffic from *P* follows the more direct path to the Internet. Second, to correct the path towards *P* the service controller initiates the following actions in order: (i) The datacenter router at location B is instructed to advertise a more specific route (i.e., a /32) to *P* via its BGP session with provider edge router *PEb*. Since routes with more specific prefixes are preferred, the path via *PEb* will be preferred over the path via *PEa* as this more specific route is distributed via the network. (ii) Once SMOG monitoring has confirmed that traffic is no longer using the path via *PEa*/*Ra* to reach *P*, *Ra* is instructed to withdraw the /29 in its advertisements, leaving the path via *PEb* as the only available path to reach *P*. (iii) Then, *Rb* in turn is instructed to advertise the shorter prefix (/29) so that we are back to the original state, except that both *P* and its associated prefix are now associated with location B.

Note that this orchestrated route control completely prevents packet loss due to the route changes since we follow a make-before-break approach whereby there is always at least one (and for some time two) valid paths to *P*. Further, very specific routes are typically

[1] The VPLS endpoints *Va* and *Vb* are in fact encapsulated within the datacenter routers (*Ra* and *Rb*), but we show them separately to simplify the exposition.

not advertised outside of the AS, i.e., the route manipulation used by SMOG would result in other PE routers within the AS following the desired path, but would not cause route convergence or instability in the larger Internet (outside the AS).

3.3 Optimized Server Placement in SMOG

While the previous section shows how SMOG can carry out migration of application servers across WANs, we still need a mechanism to decide the best position of the application servers. There are two sub-problems here. First, the service controller needs to decide *where* to place servers. This is required to ensure good service to the maximal number of clients, while taking into account metrics important to the application such as delay, loss, etc. Second, the service controller needs to decide *when* to migrate the servers. This ensures that SMOG can react, sufficiently fast, to load and network changes to avoid overload and minimize poor connection quality. At the same time, we would like to avoid inducing oscillations, and overreacting to small changes.

To address this problem in the context of network games, the service controller first collects information about the network, including the hosting locations, and the set of players on the game server. SMOG monitors the connections of the game server to determine the IP addresses of the players. It then determines the latencies the clients would experience from the various possible hosting points. This may be done using either active or passive techniques. For example, active techniques include pinging the player's end hosts to estimate RTT. (For hosts that do not respond to ping, RTT may be estimated by pinging the last router on the traceroute path to the end host.) Passive techniques include the use of latency estimation tools [32, 36].

Next, given monitoring data, the service controller must make a decision about where to place the game server. While depending on the application requirements, various metrics [37] can be used to gauge the quality of placement, in this paper, we focus on minimizing the number of online game players who experience high latency during game play i.e., whose latency to the game server is greater than a threshold latency, L_{Th}. L_{Th} can be specified by the game provider or known from evaluation studies (e.g., [3]). Using this threshold value, we formulate the *game server location problem* as a variant of the *generalized assignment problem* [38]: Suppose we are given (i) a set of hosting points, $H = \{1, 2, \ldots, n\}$, (ii) a set of servers, $S = \{1, 2, \ldots, s\}$ using SMOG, and (iii) k_i clients connected to the i^{th} server, $P_i = \{i_1, i_2, ..., i_{k_i}\}$. The optimal location of the given set of game servers is obtained as the solution to the following optimization problem:

$$\text{Minimize } C = \sum_{i=1}^{s} \sum_{j=1}^{n} x_{ij} \sum_{m=1}^{k_i} l'_{ji_m} \tag{3.1}$$

with the following constraints:

$$\sum_{j=1}^{n} x_{ij} = 1, \ \forall i \in S \tag{3.2}$$

$$\sum_{i=1}^{s} w_{ij} x_{ij} \leq c_j, \ \forall j \in H \tag{3.3}$$

$$x_{ij} \in \{0, 1\}, \ \forall i \in S, j \in H \tag{3.4}$$

where l'_{ji_m} is 1, if the latency, l_{ji_m}, of the $i_m{}^{th}$ user to the hosting point j is more than L_{Th} and 0 otherwise. w_{ij} denotes the resource requirement of server i when assigned to hosting point j (e.g., VM size, number of players, CPU usage). The above optimization problem tries to minimize the number of players whose latency to their servers is higher than L_{Th}, which is given by the cost C. x_{ij} is an indicator variable whose value is 1 if the i^{th} server is located at the j^{th} hosting point; otherwise it is equal to 0. Constraints (3.2) and (3.4) ensure that only one hosting point is used for a particular server. Constraint (3.3) ensures that the total resource requirements by the assigned servers is within the remaining capacity c_j at hosting point j.

The above optimization problem has been extensively studied, and there exist several approximation algorithms to solve it efficiently. For instance, Shmoys and Tardos [38] present an efficient 2-approximation algorithm based on LP relaxation and rounding from a fractional solution, which takes only tens of milliseconds to find a solution for instances where $|S|$ is several hundred and $|P|$ is around ten [39].[2]

SMOG solves the above problem whenever a new player joins or leaves the game (or at regular time intervals) and determines the current optimal position of the server. If the new optimal location is different from the current server location, SMOG initiates a migration using techniques discussed in §3.2. To prevent frequent oscillations, simple hysteresis techniques are applied i.e., the server is actually migrated only if the improvement in the cost function is at least a certain fraction of the current cost (e.g., this was set to 10% in Figure 2.1c).

[2]The experiments were done using a Linux machine with a AMD Athlon 2GHz CPU and GNU LP Toolkit version 4.19.

Chapter 4

Realizing SMOG in Practice

To determine the feasibility of our approach, we implemented and deployed a prototype of SMOG in a number of data centers connected to a Tier-1 ISP backbone. In this chapter, we address the various challenges encountered during this practical realization of SMOG. First, we start by describing details of our overall system deployment. We then describe the key challenge we had to address during the deployment: high downtime incurred during Xen's migration process. Finally, we describe solutions to various deployment issues that arise in practice.

4.1 System Deployment Details

We implemented SMOG on ShadowNet [40], an operational trial network and compute platform deployed on carrier-grade equipment and located within a Tier-1 ISP. Our implementation used 64bit CentOS 5.4 as many FPS games have dedicated servers which run on Linux (e.g., Quake III, CSS, Team Fortress, etc.). We used two physical servers on ShadowNet, deployed in Illinois and Texas, to act as hosting points for the game servers. These compute platforms are connected to the provider backbone via a 1Gbps link and have an average round trip time (RTT) between the two sites of 27msec. A VPLS-based VLAN is used to connect them as a single broadcast domain in the wide area. We use the Xen 3.4.3 hypervisor for the virtual machines (VM) responsible for hosting the game servers. We modified the hypervisor to incur low downtimes for wide area migration, extending Xen migration primitives [34] with the techniques described below.

The game servers are hosted on Xen VMs. Each VM is configured to send all application packets to a default gateway through which it can reach the Internet. Using BGP, a route to the VM is advertised as a /29 prefix. We use Xen migration logs to monitor the status of the migration. After the server migrates to the new location, SMOG instructs the VM to send a gratuitous ARP, to ensure the VLAN sends packets to the new physical location. DRBD [41] is used to ensure that the logical disk used by the VM on the physical machines are synchronized. It uses synchronous replication techniques to ensure that data which is

written at the *primary* location is transferred to the *secondary* location before it is committed. In our implementation, we use the physical location at which the VM is present as the primary and the physical location to which the VM would be migrated as the secondary location. This allows us to perform VM migration without requiring the VM to be paused to ensure disk consistency, reducing server downtime.

4.2 Addressing Downtime during Migration

In our early deployments, we observed that SMOG's migration interrupted the user's game experience. For example, when migrating a Quake III game server, a game client connected to it, observes a downtime of around 460 msec during migration, even with paravirtualization enabled (around 800 msec using full virtualization). While small, this downtime cannot be tolerated by online games: the Quake III client displays a *Server disconnected* message temporarily when the default Xen hypervisor is used.

To study this problem, we instrumented the Xen migration tools to micro-benchmark the sources of delays. The migration timeline, when the VM is migrated from loc_1 to loc_2, is shown in Table 4.1 (the migration process in Xen is described in §6.1). In the timeline, we note that the VM is completely unreachable to any clients from when it is suspended at loc_1 (step 2) to when it receives the first packet at loc_2 (step 6). This downtime is caused due to the following 3 mechanisms involved in Xen's migration: (a) The last iteration of copying the *remaining memory pages* and machine state from loc_1 to loc_2, occurring between steps 2 and 4. (b) Restoration of the processor, driver state etc., of the virtual machine at loc_2, occurring between steps 4 and 5. (c) Restoring network state and setting up the IP to MAC address mapping for the VM at loc_2. This occurs between steps 5 and 6. Table 4.1 shows that the key bottleneck occurs between steps 2 and 5, during which the transferred state (processor, drivers etc.) is restored at loc_2 and the VM is brought up. We noticed that a large fraction of the time between these steps is due to the hypervisor at loc_2 waiting for the hypervisor at loc_1 to close the connection used to transfer the VM state during migration. We modified Xen to remove this wait condition (Mod). With this extension, we found that downtime reduced to around 170 msec.

Although this downtime for modified Xen is very low, it can result in a few consecutive packets (3 to 4 for games like CSS and Quake III) from clients to the game server being lost. This generally goes unnoticed by game players since most online games use techniques like *dead reckoning* to deal with transient packet loss. Our user study (§5.1) shows that this downtime is in fact not noticeable by humans during gameplay. Nonetheless, some

#	Migration Step	FV	PV	Mod
1	Migration started at loc_1	0.239	0.096	0.146
2	VM suspended at loc_1	10.844	9.160	9.355
3	VM destroyed at loc_1	11.073	9.187	9.674
4	Received all pages at loc_2	11.079	9.173	9.424
5	Domain restored at loc_2	11.430	9.436	9.507
6	1st packet received by VM at loc_2	11.646	9.622	9.523
	Downtime	0.802	0.462	0.168

Table 4.1: Typical timeline (in seconds) for Xen VMs migrated from loc_1 to loc_2 for a fully-virtualized (FV) VM, paravirtualized (PV) VM, and modified Xen (Mod). Migration starts at loc_1 at time $t = 0$ sec.

games use TCP rather than UDP as transport. These games would delay recent packets until lost packets are retransmitted and, thus, would benefit from minimizing loss even during migration. For such games, simple buffering techniques at the hypervisor can be used to ensure reliable delivery of packets to the game server. We note that for most games timely delivery of recent packets is typically more important than redelivery of older packets containing stale data and thus, minimizing migration downtime is more critical as compared to reducing packet loss.

4.3 Addressing Challenges in Practice

In this section, we discuss various practical issues associated with SMOG's deployment:

Network overheads from performing migration: The overheads caused by VM migration can interfere with game play or other applications sharing the same physical infrastructure. To characterize this overhead, we perform some simple back-of-the-envelope calculations. In particular, we characterize the *amount* of overhead associated with a migration, the *duration* that overhead lasts, and the *frequency* with which migration occurs.

Amount of overhead: We first determine the amount of additional traffic sent due to a migration. In our experiments with Quake III servers (Chapter 5.2), we found that migrating a 1GB VM required sending a total of 1.008GB over the network. The results from the mshmro.com trace (Chapter 2) show that on an average a game server might have to be migrated 12 times a day to ensure it is close to the optimal location for its current users. This results in an average of 143.3 KBps traffic per game server. However, using mechanisms to reduce the network footprint of migration (e.g., Satori [42]), this can be reduced by around 90%, to as low as 14.3 KBps. This amount is significantly smaller than the traffic already generated by a standard FPS server with 32 players, which typically amounts to around

80KBps [10].

Duration of overhead: Next, we estimate the duration for which this overhead exists in the network (the amount of time required to migrate a VM). This time is determined by the rate at which VM data is transferred to the destination location and the rate at which the VM's memory is written to by applications. Our evaluation in §5.2 shows that even when the available network bandwidth for VM migration is as low as 200Mbps, under typical game loads and typical RTTs between the locations, it takes less than 30sec to complete a single migration.

Frequency of overhead: Finally, we estimate the frequency with which SMOG migrates VMs. As mentioned earlier, SMOG performs a migration only when the improvement in the cost is beyond a certain threshold. This threshold is configurable and can be set based on the load on the network. The results in §2.2 show that setting it to 10% of the current value results in just 4 migrations in a total of 7 hours. In general, the threshold can be set by the system administrator considering application requirements and the current load on the datacenters.

Cross-AS migration: With the techniques discussed in §3.2, no inter-AS advertisements are generated during the migration of a VM as long as the hosting platforms A and B are in the same AS. However, if the hosting points are located in different ASes, external routing updates may be generated. Inducing BGP routing advertisements could trigger flap damping or routing protocol reconvergence, which in turn could lead to large outages intolerable by online games. One way to address this is to have the gaming provider run its own AS (SMOG-AS), an Autonomous System that interconnects all SMOG hosting platforms. The SMOG-AS peers with each AS in which a hosting platform is present. It announces the prefixes corresponding to the game servers hosted on the SMOG framework. This ensures that no external routing updates are triggered after VM migration, as the SMOG-AS announces prefixes consistently across all peering points, regardless of its internal route changes due to specific prefix announcements. However, one disadvantage of this approach is that the SMOG-AS may need to carry all the long-distance traffic destined to the servers in SMOG hosting platforms, in case it peers with many ASes using hot potato routing. We leave the design of more sophisticated mechanisms to address the multi-AS scenario to future work.

Handling Persistent State: Our implementation uses DRBD to ensure that the persistent storage of the VM is synchronously replicated between the hosting point where the application is hosted and the hosting point to which it is migrated. However, since several hosting points can be used for the game server, synchronously replicating storage across all of them would lead to significant overhead and wastage of resources. SMOG deals with this problem by replicating data *on-demand*. This is accomplished as follows: (a) When a

server has to be migrated from loc_1 to loc_2, SMOG first starts asynchronous DRBD replication between them, (b) Once the storage devices become consistent, DRBD is switched to the synchronous mode and SMOG can initiate the migration of the VM. We note that the initial asynchronous replication in this process would increase the total time required by SMOG to perform a migration. However, since most games follow a regular daily pattern and their load is predictable (Chapter 2), the next hosting point can be pre-calculated by SMOG and the asynchronous replication can be initiated well before the actual need arises.

Synchronous replication schemes work well for games like Quake III and CSS that maintain very little persistent state. However, due to its high write latency, synchronous replication can affect the performance of games like MMOGs, which maintain persistent state (for example, inventory, health) for each player. To understand the storage performance requirements for MMOGs, we instrumented a popular open-source MMOG, Planeshift (PS) [15]. PS uses a MySQL backend to store player information. We noticed that for one player, PS sends updates to the database every 300sec (this rate is configurable) and the rate scales linearly with the number of players in the game. We believe that the low per-user data update rates and update batching seen in PS generalizes to other MMOGs because, otherwise, the database would be a bottleneck for game performance [43]. Further, some games [15, 44] can be configured to place the database on a server different from the server which hosts the game. This decoupling of the compute and storage server allows SMOG to perform the migration of the persistent storage only when necessary (for example, when the database update frequency is high), thereby reducing the overall overhead of migration. While the above heuristics try to address the problem in certain scenarios, the general question of how to migrate persistent game state during peak loads with 1000s of players online, ensuring low application downtime and overhead, is an open research question and left to future work.

Chapter 5

Evaluation of SMOG

In this chapter, we study the performance of SMOG in practice. Our evaluation is twofold. First, to understand the end-to-end performance of SMOG and its effects on end-user game experience, we conduct a user study using Quake III by simulating real world game play conditions (§5.1). Second, to understand performance issues in SMOG under varying workloads, we instrument a deployment of SMOG to directly measure the relevant performance metrics (§5.2).

5.1 User Study

SMOG aims to minimize the effect of migration on end users. To evaluate how SMOG achieves this goal, we conducted a user study using Quake III running on top of SMOG. To emulate wide-area effects, we run the user study over a LAN emulating WAN delays and bandwidth using `tc`, the Linux traffic control module.

Scenarios presented to users: We consider four different scenarios, varying the latency between the client and the server: (a) L-NM, a low latency setting with no migration, (b) LL-M, a low latency setting with migration to another low latency server, (c) H-NM, a high latency setting with no migration and (d) HL-M, a high latency setting with migration to a low latency server. Comparing the results observed for the scenarios with similar latency but with and without migration can help us understand the effects of migration. The two extreme latency conditions help us understand how users react to migration under different network conditions.

Table 5.1 shows the latencies we use for each scenario. To choose realistic values, we measured RTTs using *ping* to the top 500 Quake III servers around the world (obtained from Game Tracker [27]) from a host on the UIUC campus. Figure 5.1 shows the resulting CDF. We use the 95th percentile value (i.e., 200ms) from this distribution as the RTT between the client and the server for the high latency (H) scenarios of the user study. We use 50msec as the RTT in the low latency (L) scenarios, since it is known to be the limit above which user

	Migration?	C-S RTT before (ms)	C-S RTT after (ms)	S-S RTT (ms)
L-NM	No	50	50	NA
LL-M	Yes	50	50	27
H-NM	No	200	200	NA
HL-M	Yes	200	50	150

Table 5.1: Scenarios shown to user study participants, by client-server (C-S) latency before and after migration, and server-server (S-S) latency between hosting points.

performance degrades [33]. The RTT between the two physical servers that serve as hosting points in LL-M is set to 27msec, emulating the conditions on ShadowNet [35]. In the HL-M, this RTT is 150msec in order to be consistent with the latency from the client to the two servers. The bandwidth between hosting points is set to 465Mbps, as in ShadowNet.

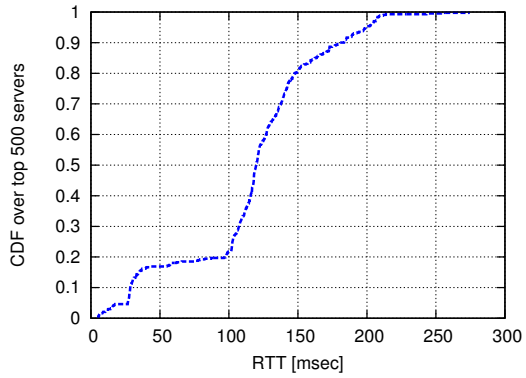
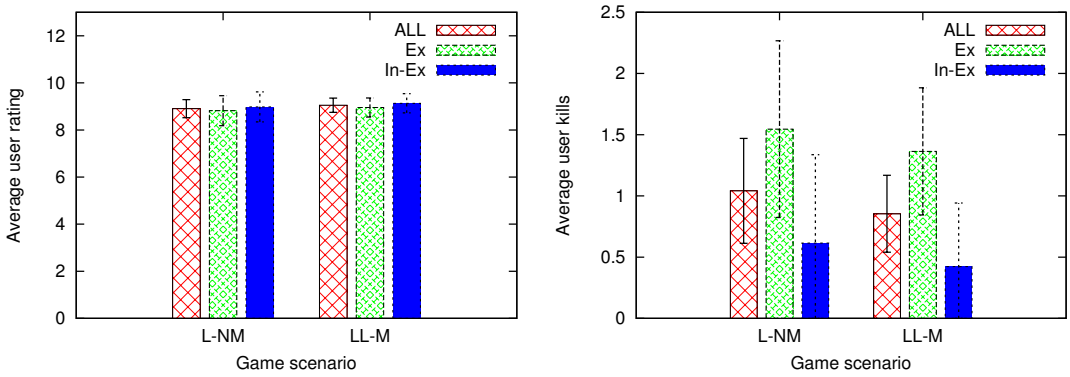


Figure 5.1: CDF of RTT to top 500 Quake Servers

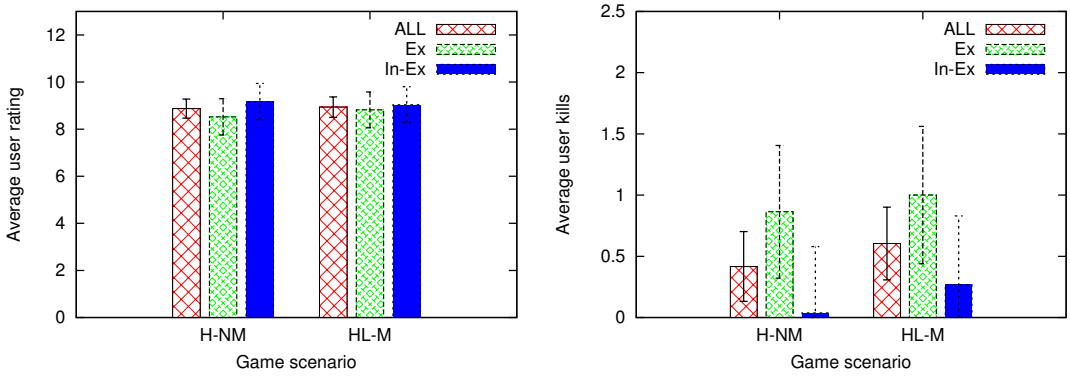
Procedure: Our user study consists of each participant playing a trial consisting of number of pairs of game scenarios, either (L-NM, LL-M) or (H-NM, HL-M). To remove any bias caused by the order in which these scenarios are played, we randomize the trial order by having each player play each trial pair twice (4 trial pairs and 8 games total). The users are told that they would be playing the game under network conditions that may differ across trials, but they are not given the specific details of the differences between them. They are also not told about the fact that some of trials can involve migration of the game server.

Before the experiment, each user first plays a practice match for 5 minutes on the `q3dm1` map of Quake III over a LAN environment. Each game round in the experiment lasts one minute. All rounds are *death match* games in which the user plays against a computer bot on the game server. We use the `q3dm1` map since it is small enough for a controlled experiment. In every round, both the human player and the bot start from fixed *spawn points*. For those scenarios with migration, the migration completes around half-way into the game. This ensures that users have (almost) equal time to play under different latencies,



(a) Average user ratings (b) Average user kill counts

Figure 5.2: User study results for low latency scenarios



(a) Average user ratings (b) Average user kill counts

Figure 5.3: User study results for high latency scenarios

before and after the migration of the server. These measures decrease the number of random factors that affect gameplay across different rounds.

Participants: We conducted a total of 96 trials of the 4 scenarios above, using a total of 24 participants. Most of the participants were already familiar with FPS games, with 95% reporting that they played FPS games before, and 50% reporting that they played FPS games regularly at least once a week at some point in their life. In this study, we regard these users as experienced. Irrespective of the participants' experience, all players perform the same tasks during the user study.

Metrics: We evaluate SMOG's user-perceived performance with the following two metrics:

(a) *User Ratings*: To provide a subjective evaluation of the game experience as observed by the users, we asked users to rate the quality of their game experience on a scale of 1-10 based on the lag experienced during the round, with 10 being no observable lag at all and 1 being completely unplayable. The main aim of using this metric is to verify if users can

consciously observe the effects of migration of the game server during the game play.

(b) *Kill Counts*: We measure the number of kills a user accomplishes in each of the rounds played. This gives us an objective way of measuring the user's performance.

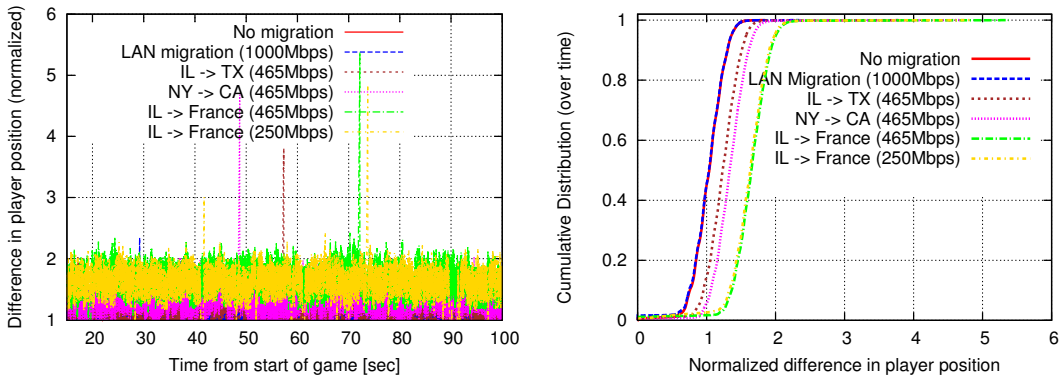
Results: Figure 5.2 and Figure 5.3 show the results of our user study using the above two metrics. For these, we divide all the users (All) into two groups: experienced (Ex) and inexperienced (In-Ex). Figure 5.2a shows the average rating given by the users, for different experience levels. The error bars indicate the 95% confidence interval. We find that the average ratings given by the users for the low latency scenarios with (LL-M) and without migration (L-NM) are quite close to each other. This shows that users cannot distinguish between the two, thereby allowing us to conclude that the *downtime due to migration in SMOG cannot be perceived by users*. We note that the same results hold for the experienced (Ex) and the inexperienced users (In-Ex), when considered separately. Figure 5.2b shows the average kill count of all the users for the low latency scenarios. This graph shows that the presence of migration does not significantly affect user performance in the game.

Figure 5.3 shows the average user rating and the average user kills for the high latency scenarios. These results are similar to those under low latency and thus, show that even under different latency conditions, SMOG's migration does not negatively affect user experience and is not consciously noticeable by a user.

5.2 Performance Analysis

To generalize our results, this section studies performance under a wider range of operating conditions. We use two metrics for evaluating SMOG's performance: (i) application downtime, which directly affects user experience in online games, and (ii) the total migration time, which measures the time during which SMOG's operations can interfere with that of the application. We note that lower level metrics such as application I/O rate etc. directly affect the above metrics and hence, we do not take them into account. All experiments were performed using a paravirtualized VM with 1GB of memory and the modified Xen hypervisor, described in Chapter 4.

Effect of downtime: During migration, since the server is unreachable for a period of time, the player's predicted position in the virtual world on the server and the player's position on the client can diverge, leading to a possible glitch in game play. Table 5.2 shows the network conditions under which we measured the downtimes experienced by the application hosted on SMOG. These are based on various real-world network parameters.



(a) Average user ratings (b) Cumulative fraction

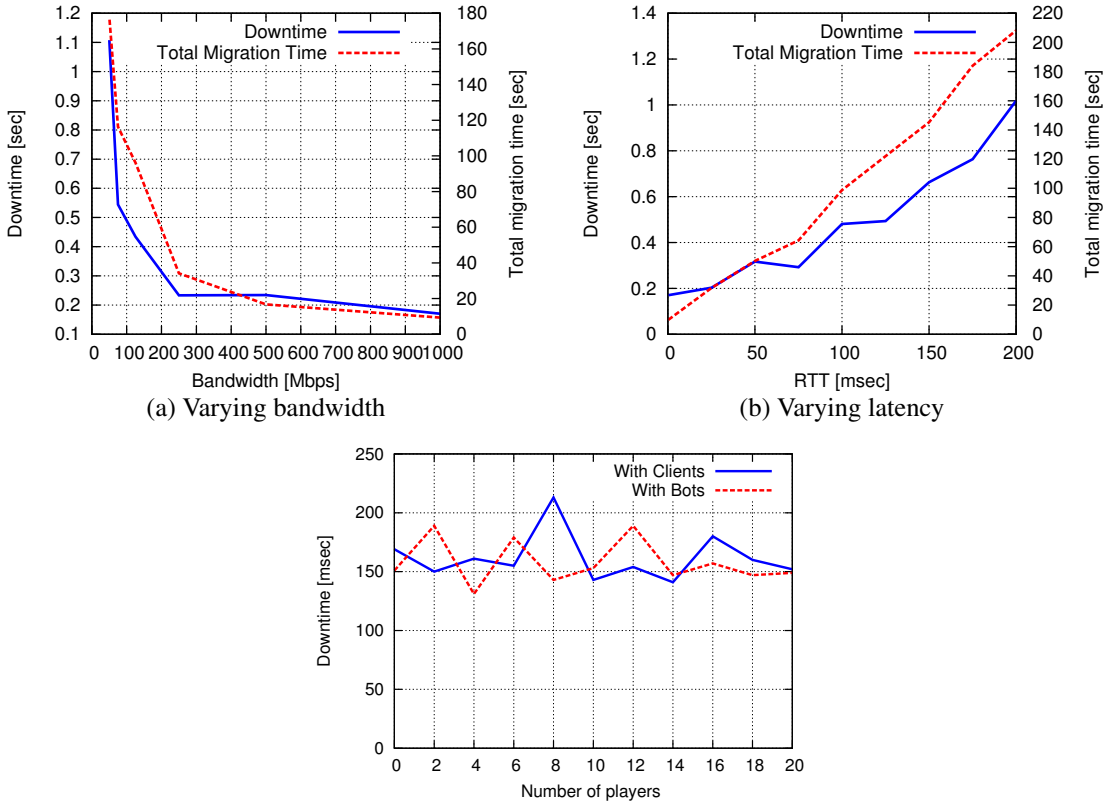
Figure 5.4: Difference in player position between server and client

Downtime (msec)	Bandwidth (Mbps)	RTT (msec)	Source/Destination pair
170	1000	0.5	LAN
220	465	27	IL to TX
240	465	45	NY to CA
350	465	100	IL to France
480	250	100	IL to France

Table 5.2: Downtime under various network conditions

Figure 5.4a shows how the difference in player position on the server and client varies over time in the game, for different values of downtime caused by migration. The differences in error are normalized with respect to the width of the *bounding box* of a player in the virtual world (i.e., the width of 1 player avatar). The figure shows that the player position error spikes only for a few milliseconds corresponding to the downtime due to migration. The increased average error for different downtimes in Figure 5.4a is due to increased network latency and does not measure the effects of migration. We observe that the maximum normalized position error is about 5. We note that this error is around the same as that observed during our user study (in cases where migration was involved) and hence, may not be noticeable by game players in practice. Figure 5.4b shows the cumulative distribution (over time) of this error. It shows that during a game lasting 100 seconds, less than 0.02% of the time, the player position error is more than twice the average showing that the effect of migration lasts for a small period of time.

Effect of bandwidth: To measure the effect of bandwidth on SMOG's performance, we varied the bandwidth available between the two hosting points using `tc`, fixing the average RTT between them at 45msec. Figure 5.5a shows how downtime and total migration time vary with bandwidth. Both metrics grow sharply with the decrease in bandwidth, and the downtime remains low (around 200 msec) even for bandwidths as low as 200Mbps. We



(a) Varying bandwidth

(b) Varying latency

(c) Varying application load

Figure 5.5: Performance of SMOG, measured using downtime and total migration, while varying different operational parameters.

observed that any downtime less than 400msec does not cause a noticeable interruption in Quake III game play. Thus, using the modified Xen migration (Chapter 4), SMOG can work seamlessly if the bandwidth available for migration is more than 125Mbps. We note that this bandwidth and latency is only needed for the last round of the migration to ensure low downtime. This bandwidth requirement can be satisfied using recent proposals such as SWAN [45] or B4 [46].

Effect of latency: Figure 5.5b shows the variation of downtime and total migration time with network RTT between hosting points. The bandwidth for all these experiments is fixed at 465Mbps (similar to the effective available bandwidth on ShadowNet [35, 47]). Although the downtime varies linearly with RTT, it increases quickly and reaches 450msec for an RTT of around 120msec. For higher RTTs, the downtime can be noticed by the users of Quake III. We note that the average RTT between the US Midwest and Western Europe (France), and Western Europe and Eastern Asia, are both around 100msec. Our results in Chapter 2 (in particular, Figure 2.1c) show that migration will most likely only occur between timezones that are close by. Thus, the modified Xen migration is sufficient

for most regular migrations.

While SMOG has sufficiently low downtime for regular network conditions, the downtime incurred can be high enough to disrupt game play, for very low bandwidths (< 125 Mbps) or very high RTTs (> 120msec) between the hosting points. However, we note that using other optimizations to Xen's migration [35], we can expect to further decrease the downtime. Using these optimizations, SMOG can work seamlessly for bandwidths as low as 50Mbps and RTTs as high as 180msec. To migrate under more extreme conditions, SMOG can perform multiple short migrations instead of one long migration. While this increases the total overhead, SMOG can select individual hops so that each migration stays below the threshold latency noticeable to end users.

Effect of application load: Increased application activity can also influence migration. For example, increased writes to memory can dirty more pages, leading to increased total migration time. To quantify this, we measured the downtime experienced by a Quake III server by varying the numbers of (a) *bots* and (b) *real clients* connected to it. We note that while the change in the application state depends on the number of bots, the change in kernel (for example, network) state depends on the number of (non-bot) clients connected to it. Hence, these two experiments help us measure the effects due to changes in different aspects of the VM's state. Figure 5.5c shows the results of this experiment when using a 465Mbps and 45msec RTT link for migration. Even with up to 20 players on the server (bots or clients), the downtime is almost the same as that with no players on it. Hence, we can conclude that for games like Quake III, increased application load does not significantly affect the user perceived experience with SMOG.

Chapter 6

Related Work

The main goal of SMOG is to provide dynamic and seamless live wide area migration to game servers in order improve player gaming experience. SMOG aims to perform these functions in an application-independent manner. SMOG builds on several previous works and is related to three main areas of research as illustrated below.

6.1 Migration of Virtual Machines

The advent of low-overhead virtualization technologies (e.g., Xen [48]) has made techniques like server migration practically feasible. The idea behind such migration is to be able to suspend the operation of the virtual machine (VM) on one physical machine and to resume it at another physical machine, while preserving the state of the whole machine (including that of the operating system, running processes, I/O devices, CPU registers etc.). Existing migration techniques can be categorized into (a) *suspend and resume*, and (b) *live* migration. Suspend and resume mechanisms like those incorporated in VMware GSX server [49] and ISR [50] suspend operation of the entire machine at the first location and then transfer the state over to the target machine. While they incorporate several optimizations like using copy-on-write disks, ballooning unused memory, compression techniques to transfer state, and using distributed file systems such as Coda [51] and AFS [52], they incur several minutes of downtime during which the server becomes unavailable. Hence, such mechanisms are unsuitable for applications like interactive online games where every second counts.

On the other hand, live migration techniques like those used in Xen [34] and current versions of VMware [53], strive to achieve sub-second downtime. Xen uses paravirtualization and a *pre-copy* mechanism to achieve this – during the pre-copy phase, the dirty pages of the VM's memory are iteratively transferred to the destination while the VM is allowed to run. During the last phase, the VM is completely suspended and the remaining state is copied over. By ensuring that the last phase has to deal with only a few memory pages, Xen achieves a downtime of few tens of milliseconds. However these mechanisms assume

that the physical machines are on a single LAN and hence cannot be directly adopted in our wide area settings. In this work, we leverage the live migration primitives provided by Xen and provide network support in order to enable wide area server migration.

Various approaches to wide area server migration using network level support have been proposed earlier. Openflow [12] facilitates server migration through the use of Openflow enabled swicthes. It allows the modification of switch entries in order to ensure reachability after the migration finishes. Unlike Openflow, SMOG works with legacy network components. Several approaches use such network level redirection to facilitate wide area live migration of VMs. Bradford et. al. [14] propose one such mechanism which uses dyn-DNS with tunneling, along with pre-copying and write throttling, to migrate servers across WANs but experiences several tens of seconds of downtime. Transit portal (TP) [13] is another such mechanism which provides a technique to perform wide area server migration using wide area route control to divert traffic to the new server location. However, such an approach can result in outages for several seconds to minutes (equal to the convergence period of BGP in the wide area) along with termination of existing connections. Contrary to these mechanisms, we leverage VLAN technologies to minimize outages to less than a second and provide seamless connectivity to the end-users.

SMOG builds on existing mechanisms [54, 55], using VPLS and VLANs to support seamless WAN migration of VMs. It performs route control to reap the benefits offered by such migration. Wide area route control enables us to achieve least latency paths from the clients to the servers hosted in the cloud.

6.2 Latency Sensitivity of Online Games

The requirement of having low and predictable latency between the game server and the player to ensure good user experience is well known, both in the game research community and the online gaming industry. Several studies have tried to evaluate how player performance and satisfaction vary with the network Quality of Service (QoS) experienced by players during the game. Chen et. al. [3] study the effect of QoS on MMOGs and conclude that the player session times decrease with increased network latency and loss rate. A study [4] of how latency affects player decisions to choose game servers, shows that players of Quake III do not typically choose servers whose latency to them is more than 150ms. Another study [5] examines real world servers of Unreal Tournament 2003 [26] to understand the typical ranges of packet loss and latency and concludes that while packet loss may not affect player performance significantly, latencies above 100ms can significantly

degrade performance of shooting precision weapons and latencies above 150ms make the game *feel* sluggish. Further, Wattimena et. al. [6] develop a metric to quantify player satisfaction and determine how this varies with network conditions – they infer that increased latency and jitter decrease player satisfaction.

The above studies along with several others [16–19] provide similar conclusions – network QoS has a very significant impact on player performance and satisfaction; both for fast-paced FPS games and the slower MMOGs. Drawing motivation from this large body of work, SMOG tries to ensure that players achieve high levels of satisfaction by ensuring that the network QoS requirements are always met – otherwise, it migrates the game server to a more suitable location given the current set of players.

6.3 Efficiently Hosting Online Games

Given the strict QoS requirements of online games as discussed above, game providers find it challenging to host interactive games over the Internet. Large-scale games such as Second Life [20] have replicated server infrastructure across the Internet which ensures that players can find servers that are close to them and hence, achieve lower latency. Others such as Quake III [8] are hosted by individual players which ensures that the servers are distributed all over the world. Further, games like Halo [56] use matchmaking services [32] to ensure that the users are matched to the servers that are closest to them. However, these solutions do not solve the problem of hosting games which have not yet achieved sufficient player population, or have players who play on distant servers due to reasons other than network latency (e.g., social preferences).

Several solutions for simplifying the problem of hosting games have been proposed in the research community. Many of these [57–59] include the use of middleware platforms which focus on dynamically increasing the number of servers required to deal with client load. These approaches, hence, help in determining *how many* servers are needed to handle player load. Beskow et al. [60] use *middleware redirection* to achieve migration of game state in the wide area, but this can suffer from large downtimes during migration of game state. Further, some proposals dynamically improve player latencies by changing the way game software is architected – for example, by using proxies [60] or P2P infrastructures [9].

In contrast to these approaches, SMOG helps to determine *when* and *where* to move game servers in order to reduce network latency and achieve a better user experience. SMOG leverages network layer support to achieve this while ensuring very low application downtime. Further, we believe the VM platform approach taken by SMOG is a more prac-

tical solution because it relieves game developers from additional software and operational complexity. SMOG is more practical to deploy at scale because it requires only infrastructure and primitives that most cloud service providers already have in place. In contrast to previous solutions to improve latency, existing latency-sensitive games could benefit from SMOG without any changes, thus providing an immediate potential customer-base for such providers.

We note that SMOG migrates game servers in response to variations in server loads and connection quality in the wide area. SMOG does not deal with the unpredictability of network characteristics within a single cloud data-center which has been demonstrated widely [61–63] and can impair the ability to host latency-sensitive applications. Several recent works such as SecondNet [64], Oktopus [65] and D3 [66] try to solve this issue and provide mechanisms to ensure that applications running in cloud settings can achieve predictable performance. These mechanisms complement SMOG and can help in building a comprehensive game hosting framework.

Chapter 7

Conclusions

Today, online games are one of the most popular form of entertainment with a multi-billion dollar market [1] and a rapidly increasing player population [67]. In spite of their growing popularity, game providers encounter several difficulties in hosting their games due to their strict requirements on network Quality of Service (QoS). In addition, several games may not have the required player population to justify their need for a dedicated infrastructure or may require the game servers to be hosted by individual players. This can lead to further poor connectivity to the game servers. Cloud computing platforms provide an attractive option for such games as they reduce the cost of hosting games while eliminating issues such as cluster maintenance. However, given the requirements of online games, such platforms can provide no guarantees on the QoS achieved between the players and the game servers. In this work, we aim to solve this problem of hosting online games on cloud platforms while ensuring good network QoS to the players and hence, high user satisfaction. To this end, we propose a platform for Seamless Migration of Online Games or SMOG, which ensures that the location of the game server is dynamically chosen to provide the best game play to all the players on the server.

Unlike traditional cloud computing platforms, SMOG's infrastructure is distributed across the wide area and integrated with network support to provide enhanced service to online games. SMOG builds on traditional live migration techniques and uses them along with intelligent route control in the wide area to seamlessly migrate game servers across datacenters. SMOG's control platform continuously monitors connections from a game server to the players. Using this as input to an optimization framework, it determines the best physical location for hosting the server while meeting the capacity constraints of the endpoints. Once the control platform determines where to move the game server, SMOG *live migrates* the game server to the target location. It then creates ARP and BGP announcements in order to ensure that the traffic from the players reach the game server along the best paths. Using these mechanisms, SMOG ensures that games achieve better end user experiences.

We realized a working prototype of SMOG on a Tier-1 ISP's backbone network, achiev-

ing less than 200 ms of application downtime during migration. For this, we modified the default hypervisor of Xen to speed up the connection close between the end points at the end of migration. This change decreased the downtime from about 470 ms to just 170 ms, ensuring uninterrupted game play for the users. Further, we evaluated our prototype of SMOG using a user study, where users were made to play various scenarios, with and without migration, and asked to rate the game play based on their experience. We found that users did not notice the presence of migration and that their game play (quantified using number of kills) was not affected by it. This shows that SMOG achieves seamless server relocation while being unnoticed by the players. Further, we show that SMOG is capable of operating under a variety of network conditions and can tolerate bandwidths as low as 50Mbps and network RTTs as high as 180ms.

7.1 Future Work

While this work focuses on online games, SMOG can also be used to improve the performance of other latency-sensitive applications and online services. It is a first step towards addressing the broader challenge of integrating cloud resources with the network, an approach that we believe is essential to realize the full potential of cloud computing for highly-interactive applications. The following include avenues for future work which would enable a comprehensive framework for hosting such applications on cloud platforms.

Persistent state migration: Some games, such as Massively Multiplayer Online Games (MMOGs), maintain long-lived persistent game worlds on servers. While the state replication mechanisms used by SMOG work well for such games under low update rates, it is unclear how the use of DRBD would scale under (a) increased load, with several 1000s of updates per second and (b) increased migration rate, with the geographical distribution of player population on a server changing every few minutes. Using other state replication techniques such as wide-area SAN present a promising alternative to using synchronous replication mechanisms like DRBD.

Supporting multiple servers: Games like Second Life [68] host a single virtual world on a distributed collection of servers. Today, these servers are always hosted within the same data center but SMOG would allow them to move to geographically distributed data centers. Extending SMOG to support migration of multiple servers while ensuring low overhead and consistency, is an interesting problem and will allow such games to reap of benefits of wide area migration.

Bandwidth and other resource constraints: In addition to latency, emerging game

hosting platforms also have substantial bandwidth constraints. For example, OnLive [11] streams game state to players as a video rather than as small state updates. Integration of such hosting services with SMOG could substantially reduce bandwidth costs and improve interaction latencies by ensuring that videos are always streamed from servers near clients. The optimization framework of SMOG can be extended to include bandwidth constraints to address this problem.

Support for other interactive applications: In this work, we describe SMOG as a platform for efficiently hosting online games. However, its main primitive of seamlessly migrating application processes across the wide area can also provide similar benefits for other latency sensitive applications such as video conferencing etc. It would be interesting to understand the performance of SMOG when used for such applications as they might have different metrics to optimize along with a different set of constraints (e.g., bandwidth available for clients).

New network primitives: SMOG provides seamless wide area migration as a primitive for highly interactive online games. We believe that exposing other network-level properties and providing new network primitives to the applications can help them perform better. While such exposure can lead to unintended problems (e.g., new security vulnerabilities), they will nevertheless improve end-user experience as applications can decide what support they require from the underlying network.

References

[1] "Online game market," http://www.gamesbrief.com/2010/06/the-online-games-market-was-worth-15-billion-in-2009-and-will-grow-to-20-billion-in-2010/.

[2] "217 million people play online games," http://techcrunch.com/2007/07/10/217-million-people-play-online-games/.

[3] K.-T. Chen, P. Huang, and C.-L. Lei, "How sensitive are online gamers to network quality?" *Commun. ACM*, vol. 49, no. 11, 2006.

[4] "Lag over 150 milliseconds is unacceptable," http://gja.space4me.com/things/quake3-latency-051701.html.

[5] T. Beigbeder, R. Coughlan, C. Lusher, J. Plunkett, E. Agu, and M. Claypool, "The effects of loss and latency on user performance in unreal tournament 2003," in *NetGames*, 2004.

[6] A. F. Wattimena, R. E. Kooij, J. M. van Vugt, and O. K. Ahmed, "Predicting the perceived quality of a first person shooter: the quake iv g-model," in *NetGames*, 2006.

[7] "Counter-strike: Source on steam," http://store.steampowered.com/css.

[8] "Quake III Arena," http://www.quake3arena.com/.

[9] A. Bharambe, J. Pang, and S. Seshan, "Colyseus: a distributed architecture for online multiplayer games," in *NSDI*, 2006.

[10] A. Bharambe, J. R. Douceur, J. R. Lorch, T. Moscibroda, J. Pang, S. Seshan, and X. Zhuang, "Donnybrook: enabling large-scale, high-speed, peer-to-peer games," *SIGCOMM CCR*, vol. 38, no. 4, 2008.

[11] "OnLive," http://www.onlive.com.

[12] "Open Flow Demo," http://www.openflowswitch.org/wp/2008/12/video-of-mobile-vms-demo/.

[13] V. Valancius, N. Feamster, J. Rexford, and A. Nakao, "Wide-area route control for distributed services," in *USENIX ATC*, 2010.

[14] R. Bradford, E. Kotsovinos, A. Feldmann, and H. Schiöberg, "Live wide-area migration of virtual machines including local persistent state," in *VEE*, 2007.

[15] "Planeshift," http://www.planeshift.it/.

[16] G. Armitage, "An experimental estimation of latency sensitivity in multiplayer quake 3," in *ICON*, 2003.

[17] K.-T. Chen, P. Huang, G.-S. Wang, C.-Y. Huang, and C.-L. Lei, "On the Sensitivity of Online Game Playing Time to Network QoS," in *Infocom*, 2006.

[18] M. Oliveira and T. Henderson, "What online gamers really think of the Internet?" in *NetGames*, 2003.

[19] T. Yasui, Y. Ishibashi, and T. Ikedo, "Influences of Network Latency and Packet Loss on Consistency in Networked Racing Games," in *NetGames*, 2005.

[20] "Second Life," http://secondlife.com/.

[21] "World of Warcraft," http://us.battle.net/wow/en/.

[22] "World War II Online," http://www.battlegroundeurope.com/.

[23] "PlanetSide," http://www.planetside-universe.com/.

[24] "Cheats could ruin online gaming," http://www.cbsnews.com/stories/2002/12/09/tech/main532309.shtml.

[25] "Steam Game Statistics," http://steampowered.com/status/game_stats.html.

[26] "Unreal Tournament," http://www.unrealtournament.com/.

[27] "Game tracker," http://www.gametracker.com/.

[28] "QStat," http://www.qstat.org/.

[29] "Mshmro.com Counter-Strike trace," http://www.thefengs.com/wuchang/work/cstrike/.

[30] "IpInfoDB," http://ipinfodb.com/.

[31] T. Benson, A. Akella, and D. A. Maltz, "Network traffic characteristics of data centers in the wild," in *IMC*, 2010.

[32] S. Agarwal and J. R. Lorch, "Matchmaking for online games and other latency-sensitive p2p systems," in *SIGCOMM*, 2009.

[33] M. Claypool and K. Claypool, "Latency and player actions in online games," *Commun. ACM*, vol. 49, no. 11, 2006.

[34] C. Clark, K. Fraser, S. Hand, J. G. Hansen, E. Jul, C. Limpach, I. Pratt, and A. Warfield, "Live migration of virtual machines," in *NSDI*, 2005.

[35] T. Wood, K. K. Ramakrishnan, P. Shenoy, and J. van der Merwe, "Cloudnet: dynamic pooling of cloud resources by live wan migration of virtual machines," in *VEE*, 2011.

[36] K. P. Gummadi, S. Saroiu, and S. D. Gribble, "King: Estimating latency between arbitrary Internet end hosts," in *IMW*, 2002.

[37] A. Karaman and H. Hassanein, "Core-selection algorithms in multicast routing - comparative and complexity analysis," *Computer Communications*, vol. 29, no. 8, 2006.

[38] D. Shmoys and E. Tardos, "An approximation algorithm for the generalized assignment problem," *Mathematical Programming*, vol. 62, pp. 461–474, 1993.

[39] H. A. Alzoubi, S. Lee, M. Rabinovich, O. Spatscheck, and J. V. der Merwe, "Anycast CDNS Revisited," in *WWW*, 2008.

[40] X. Chen, Z. Morley, M. Jacobus, and V. Merwe, "Shadownet: A platform for rapid and safe network evolution," in *Usenix ATC*, 2009.

[41] "DRBD," http://www.drbd.org/.

[42] G. Miłós, D. G. Murray, S. Hand, and M. A. Fetterman, "Satori: enlightened page sharing," in *USENIX ATC*, 2009.

[43] K. Zhang, K. Bettina, and A. Denault, "Persistence in massively multiplayer online games," in *NetGames*, 2008.

[44] "Ryzom," http://www.ryzom.com/en/.

[45] C.-Y. Hong, S. Kandula, R. Mahajan, M. Zhang, V. Gill, M. Nanduri, , and R. Wattenhofer, "Achieving High Utilization with Software-Driven WAN," in *SIGCOMM*, 2013.

[46] S. Jain, A. Kumar, S. Mandal, J. Ong, L. Poutievski, A. Singh, S. Venkata, J. Wanderer, J. Zhou, M. Zhu, J. Zolla, U. Hölzle, S. Stuart, and A. Vahdat, "B4: Experience with a Globally-Deployed Software Defined WAN," in *SIGCOMM*, 2013.

[47] B. Cully, G. Lefebvre, D. Meyer, M. Feeley, N. Hutchinson, and A. Warfield, "Remus: high availability via asynchronous virtual machine replication," in *NSDI*, 2008.

[48] "Xen," http://www.xen.org.

[49] C. P. Sapuntzakis, R. Chandra, B. Pfaff, J. Chow, M. S. Lam, and M. Rosenblum, "Optimizing the migration of virtual computers," *SIGOPS Oper. Syst. Rev.*, vol. 36, pp. 377–390, 2002.

[50] M. Kozuch and M. Satyanarayanan, "Internet Suspend/Resume," in *WMCSA*, 2002.

[51] M. Satyanarayanan, J. J. Kistler, P. Kumar, M. E. Okasaki, E. H. Siegel, and D. C. Steere, "Coda: A Highly Available File System for a Distributed Workstation Environment," *IEEE Trans. Comput.*, vol. 39, no. 4, Apr. 1990.

[52] J. H. Howard, M. L. Kazar, S. G. Menees, D. A. Nichols, M. Satyanarayanan, R. N. Sidebotham, and M. J. West, "Scale and Performance in a Distributed File System," *ACM Trans. Comput. Syst.*, vol. 6, no. 1, pp. 51–81, 1988.

[53] M. Nelson, B. H. Lim, and G. Hutchins, "Fast transparent migration for virtual machines," in *USENIX ATC*, 2005.

[54] K. K. Ramakrishnan, P. Shenoy, and J. Van der Merwe, "Live data center migration across WANs: a robust cooperative context aware approach," in *INM*, 2007.

[55] T. Wood, P. Shenoy, A. Gerber, K. K. Ramakrishnan, and J. Van der Merwe, "The case for enterprise-ready virtual private clouds," in *HotCloud*, 2009.

[56] "Halo waypoint," http://www.halowaypoint.com/en-us/.

[57] D. Saha, S. Sahu, and A. Shaikh, "A service platform for on-line games," in *NetGames*, 2003.

[58] A. Shaikh, S. Sahu, M. Rosu, M. Shea, and D. Saha, "Implementation of a service platform for online games," in *NetGames*, 2004.

[59] P. Quax, J. Dierckx, B. Cornelissen, G. Vansichem, and W. Lamotte, "Dynamic server allocation in a real-life deployable communications architecture for networked games," in *NetGames*, 2008.

[60] P. B. Beskow, K.-H. Vik, P. Halvorsen, and C. Griwodz, "Latency reduction by dynamic core selection and partial migration of game state," in *NetGames*, 2008.

[61] J. Schad, J. Dittrich, and J.-A. Quiané-Ruiz, "Runtime measurements in the cloud: observing, analyzing, and reducing variance," in *VLDB*, 2010.

[62] A. Iosup, N. Yigitbasi, and D. Epema, "On the performance variability of production cloud services," in *CCGRID*, 2011.

[63] M. Zaharia, A. Konwinski, A. D. Joseph, R. Katz, and I. Stoica, "Improving mapreduce performance in heterogeneous environments," in *OSDI*, 2008.

[64] C. Guo, G. Lu, H. J. Wang, S. Yang, C. Kong, P. Sun, W. Wu, and Y. Zhang, "Secondnet: a data center network virtualization architecture with bandwidth guarantees," in *Co-NEXT*, 2010.

[65] H. Ballani, P. Costa, T. Karagiannis, and A. Rowstron, "Towards predictable datacenter networks," in *SIGCOMM*, 2011.

[66] C. Wilson, H. Ballani, T. Karagiannis, and A. Rowtron, "Better never than late: meeting deadlines in datacenter networks," in *SIGCOMM*, 2011.

[67] "Gaming Market - Global Industry Analysis, Size, Growth, Share and Forecast," http://www.transparencymarketresearch.com/global-gaming-market.html.

[68] "Second Life Working Group," http://wiki.secondlife.com/wiki/Architecture_Working-_Group.